MIROSLAV PAULÍK

RUBY STORIES V

PRIMEHAMMER.COM

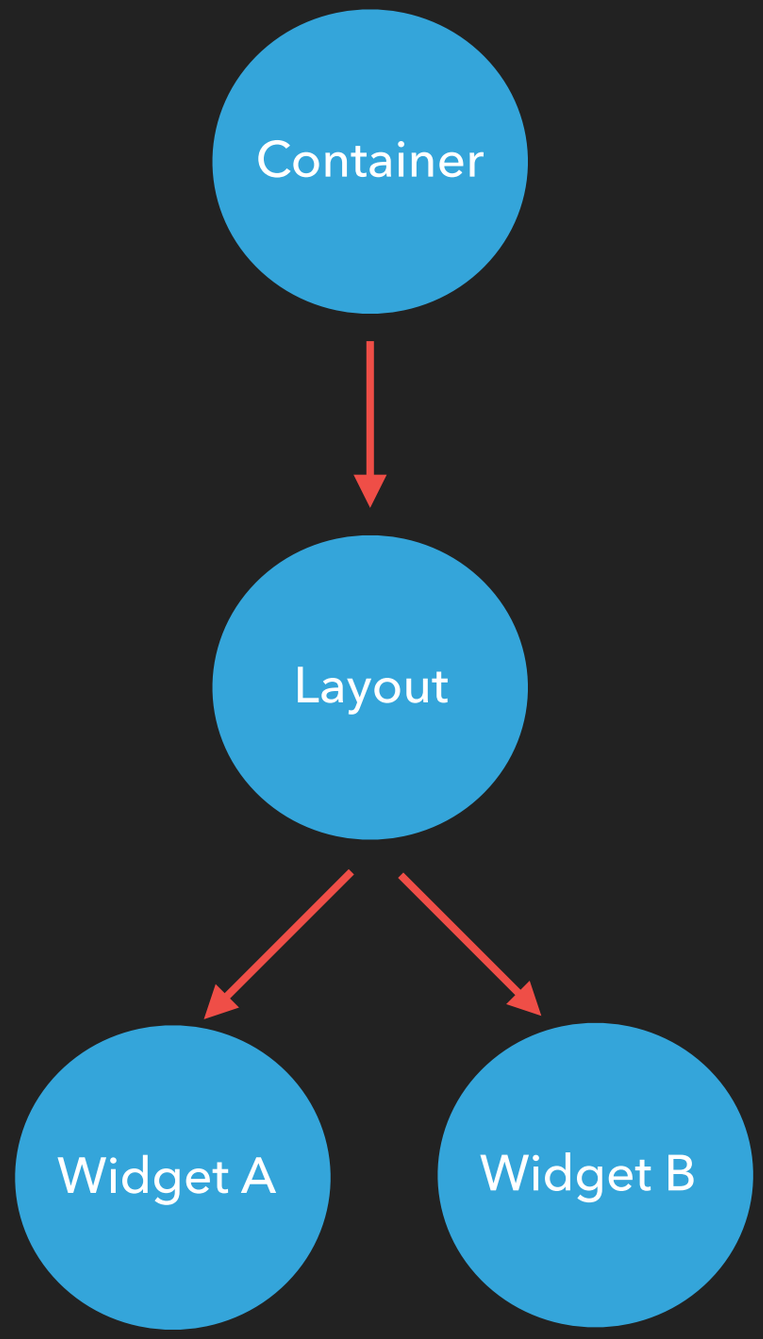
HOW TO PASS PROPS IN LARGE COMPONENT TREES

HOW GOOD IS YOUR CODE?

- Does is meet all the requirements from the task
- Is it covered by tests?
- Is it easy to read, easy to understand?
- Is it maintainable?

COMPONENT TREES IN REACT APPS

- Container
- Layout
- Widget
- Low level components like button, input etc



LARGE COMPONENT TREES

- Page container
- Layout (with tabs)
- Tab
- Widget
- Modal
- Form
- …

TASK: ADD A VALUE FROM REDUX TO A WIDGET – CALC

- Assume 3 usages per component test

- Container: 3 usages (connect, PropType, render)
- Layout (and each other component between Container and Widget): 5 usages (PropType, render and 3 in tests)
- Widget: 4+ usages (PropType and 3+ in tests)

- 3 level component tree: 3 + 1*5 + 4 = 12 usages
- 4 level component tree: 3 + 2*5 + 4 = 17 usages
- 5 level component tree: 3 + 3*5 + 4 = 22 usages
- 6 level component tree: 3 + 4*5 + 4 = 27 usages

WHAT TOOLS REACT OFFERS

- Pass props from each parent down the whole tree
- Context

CONTEXT

- Second “props” for components
- Provides a way to share values between components without having to explicitly pass a prop through every level of the tree
- Introduced in React 16, improved in React 16.3

CONTEXT – EXAMPLES

```jsx
import React, { createContext } from 'react';

export const ThemeContext = React.createContext(
  'dark' // default value
);

function Container(props) {
  return (
    <ThemeContext.Provider value={props.theme}>
      <Layout {...props} />
    </ThemeContext.Provider>
  );
}

function Widget(props) {
  return (
    <ThemeContext.Consumer>
      {theme => <button className={theme}>Themed button</button>}
    </ThemeContext.Consumer>
  );
}
```

CONTEXT – PROS

- Provided at top level component
- Consumed only where we want (Widget)

CONTEXT – CONS

- Components in the middle (for example Layout) do not provide this Context - tests will fail (or show warnings)
- More Contexts make developer's life more complicated

```
function ContainerWithMoreContexts(props) {
  return (
    <FirstWidgetContext.Provider value={props.theme}>
      <SecondWidgetContext.Provider value={props.theme}>
        <ThirdWidgetContext.Provider value={props.theme}>
          <Layout {...props} />
        </ThirdWidgetContext.Provider>
      </SecondWidgetContext.Provider>
    </FirstWidgetContext.Provider>
  );
}
```

SMALL RECAP

- We want to:
  - Just to add a value into the Widget and set it the Container
- We don’t want to:
  - Modify anything else

HOW TO KEEP COMPONENTS IN THE MIDDLE STABLE

- We tried to skip these components with Context, but we failed
- We can try to pass just one prop that will contain all other props (encapsulate all other props)

EXAMPLE 1/3

```
const mapStateToProps = (state) => ({
  products: state.allProducts,
  investors: state.allInvestors,
  price: state.estimatedPrice,
});
```

```
const mapStateToProps = (state) => ({
  products: selectAllProducts(state),
  investors: selectAllInvestors(state),
  price: selectEstimatedPrice(state),
});
```

```
const getWidgetProps = (investors, products, price) => ({ investors, products, price });

export const selectWidgetProps = createSelector(
  selectAllProducts,
  selectAllInvestors,
  selectEstimatedPrice,
  getWidgetProps
);

const mapStateToProps = state => ({
  widgetProps: selectWidgetProps(state),
});
```

EXAMPLE 2/3 – CONTAINER

```
class Container extends React.Component {
  static propTypes = {
    products: PropTypes.array.isRequired,
    investors: PropTypes.array.isRequired,
    price: PropTypes.number.isRequired,
  };

  render() {
    return <Layout investors={this.props.investor} products={this.props.products} price={this.props.price} />;
  }
}
```

```
class Container extends React.Component {
  static propTypes = {
    widgetProps: PropTypes.object.isRequired,
  };

  render() {
    return <Layout widgetProps={this.props.widgetProps} />;
  }
}
```

EXAMPLE 3/3 – LAYOUT

```
class Layout extends React.Component {
  static propTypes = {
    products: PropTypes.array.isRequired,
    investors: PropTypes.array.isRequired,
    price: PropTypes.number.isRequired,
  };

  render() {
    return <Widget investors={this.props.investor} products={this.props.products} price={this.props.price} />;
  }
}
```

```
class Layout extends React.Component {
  static propTypes = {
    widgetProps: PropTypes.object.isRequired,
  };

  render() {
    return <Widget {...this.props.widgetProps} />;

    // OR
    const { products, investors, price } = this.props.widgetProps;
    return <Widget products={products} investors={investors} price={price} />;
  }
}
```

HOW TO ADD NEW PROP NOW?

```
const getWidgetProps = (investors, products, price) => ({ investors, products, price });

export const selectWidgetProps = createSelector(
  selectAllProducts,
  selectAllInvestors,
  selectEstimatedPrice,
  getWidgetProps
);
```

```
const getWidgetProps = (investors, products, price, currency) => ({ investors, products, price, currency });

export const selectWidgetProps = createSelector(
  selectAllProducts,
  selectAllInvestors,
  selectEstimatedPrice,
  selectCurrency,
  getWidgetProps
);
```

RESULTS

- Constant number of changes
- Less naming collisions
- Less code

MAINTAINABLE CODE MATTERS

THANK YOU

OTHER APPROACHES

- Reduce depth of the component tree
- Send down whole components instead of data